The described control program dynamically schedules the operational activities performed by the IBM 9020 *multiprocessing system. Scheduling is based on program execution requirements and allows dynamic switching of Computing Element assignments.*

Storage resources are dynamically allocated by the control program to guard against the mutual interference of concurrent operations.

The trace capability of the control program is described because of its importance to the checkout and evaluation of multiprocessor systems.

An application-oriented multiprocessing system

III Control program features

by J. A. Devereaux

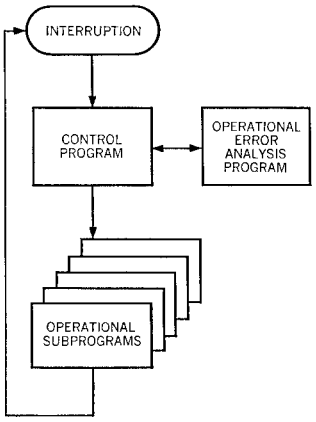
In designing a control program for the IBM 9020 system discussed in Part II of this paper, the main programming challenge was to provide for the simultaneous operation of two or more Computing Elements (processing units) without having one interfere with the others. The obvious approach, that of preassigning specific functions to each Computing Element, is too inflexible. In fact, any scheme based on the habitual notion that Computing Elements should be the focal point of control is apt to lead to inflexibilities.

In real-time applications such as the Federal Aviation Administration's National Airspace System for enroute air traffic control,[1] short response times are coupled with stringent reliability requirements to preclude a permanent assignment of specific jobs to specific Computing Elements. An approach that permits dynamic scheduling of Computing Element operations is more appropriate for the requirements of such a system. Accordingly, the design of the control program for the National Airspace System is based on the concept that programs are the focal point of control and that Computing Elements are allocated to programs according to the needs of the application.[2]

organization

To implement this concept of operation, each application program is broken down into units called *subprograms*. Each subprogram is assigned an operational priority, and the priority itself is represented by an appropriate entry in the control program's scheduling table. If a subprogram is re-entrant — i.e., simultaneously available to more than one Computing Element — it occupies multiple entries in the table, one for each instance of execution.

Figure 1 Relationship of NAS programs



Whenever a subprogram requires an instance of execution, an available Computing Element is allocated according to the subprogram's priority. The appropriate scheduling table entry is updated to reflect the allocation. With the aid of its scheduling table, the control program ensures that the subprogram operations of a given Computing Element do not conflict with each other or with the operations of other Computing Elements.

Organizationally, the entire monitoring and control function is accomplished by two system programs: the control program and the Operational Error Analysis Program (OEAP). Only the multiprocessing features of the control program are discussed here, whereas the OEAP is described in detail in Part IV. The schematic in Figure 1 shows the general relationship among system and application programs.

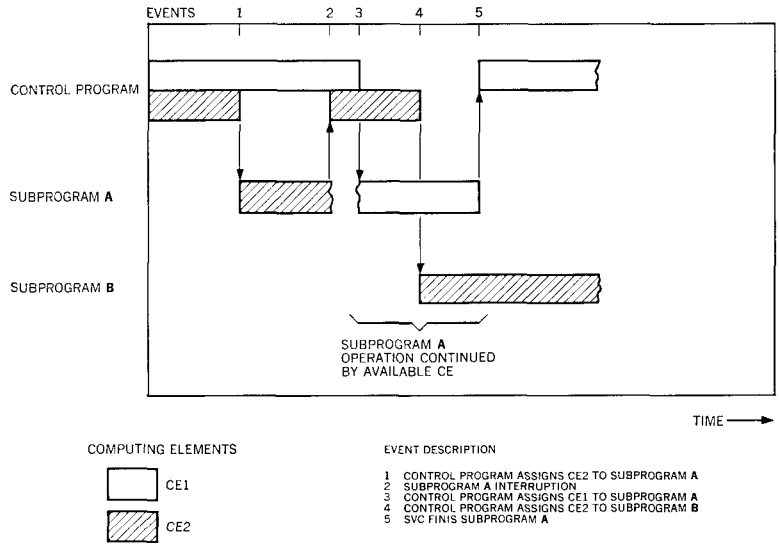
flow of control

In general, the control program gains control of a Computing Element as the result of an interruption in a subprogram. After appropriate action is taken by the control program, control is either returned to the interrupted subprogram or to another subprogram, depending upon the type of interruption and subprogram priority. Error indications are passed on to the Operational Error Analysis Program, which identifies the malfunctioning element and passes its identity back to the control program to effect system recovery and resume normal processing. The approach to system recovery is fairly straightforward; if an element has failed, it is replaced with a redundant element through reconfiguration control. During normal system operation, checkpoints are established and critical data are stored on magnetic tape. In the course of a recovery operation, data for the last checkpoint are used to reload suspect Storage Elements. To avoid possible confusion, only one Computing Element is active during the recovery process; others are held in the wait state until normal operation has been resumed.

control program re-entrance

During subprogram execution, all interruptions are *enabled* (permitted to occur) and the subprograms operate in the problem state. The interrupt-service routines of the control program, on the other hand, are executed with interruptions disabled. Thus, once interrupted, a given Computing Element will not be interrupted again until the current interruption is processed. Although this limits the number of possible simultaneous instances of control program execution to one per Computing Element, the possibility of simultaneous instances of control program execution by different Computing Elements places other restrictions on the design of the control program. The principal restriction is that routines that are not re-entrant, as well as routines that access common control data, must be protected against overlapped execution. This protection is provided with the aid of the TEST AND SET instruction. If the code or data related to the TEST AND SET instruction are not available (i.e., are already allocated for control program execution by another Computing Element), the waiting Computing Element loops until the desired code or data are available. To preclude a hung system, countdowns are built into the TEST AND SET loops,

Figure 2 Dynamic scheduling



allowing a loop to be bypassed after a reasonable delay. This ensures that Computing Elements will eventually reach a stage where external interruptions are enabled so that an element failure can be detected.

scheduling

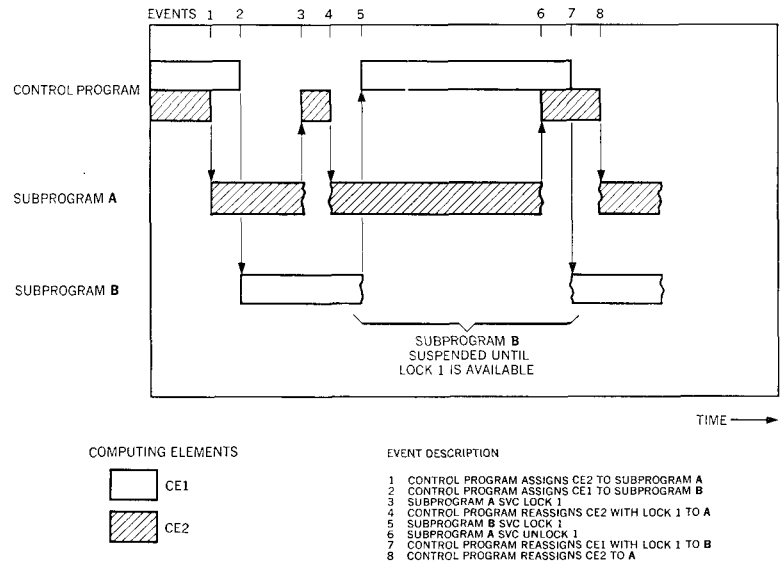
The operation of the system, as described thus far, is similar to that observed in any single processor system using multiprogramming techniques. However, there is a more interesting aspect of the operation with multiple Computing Elements. While an interrupted Computing Element is executing one of the interrupt routines of the control program, an idle Computing Element can be allocated to continue the execution of the interrupted subprogram. Potentially, this form of dynamic scheduling can reduce the overall execution time for the subprogram. An example of this is suggested by Figure 2.

dynamic storage allocation

Overlapped operations, in general, imply a requirement for central control of all common storage resources. One of the principal implications of this requirement for subprogram design is that subprograms must use the storage allocation services of the control program.

Dynamic storage allocation is implemented by SVC, the supervisor-call instruction, which generates a distinctive interruption. During its execution, let us say, a subprogram needs a particular storage resource. At this point, the subprogram identifies its requirements to the control program with the aid of SVC. If the storage resource is available, the control program allocates the resource and returns Computing Element control to the subprogram. When the subprogram no longer requires the allocated storage resource, it identifies the resource as available—again communicating

Figure 3 Dynamic storage allocation



with the aid of SVC. If a requested storage resource is already in use, the operation of the subprogram is suspended until the resource becomes available. An example of this is shown in Figure 3.

Three types of storage resources—areas, blocks, and lines—are defined:

Areas. An area is a portion of storage containing data or code that are common to two or more subprograms and that cannot be accessed simultaneously without potential loss of information. Areas are locked and unlocked by subprograms.

Blocks. A block is a section of contiguous storage within a general storage pool. The general storage pool is subdivided into subpools, each subpool consisting of a specified number of fixed-size blocks. Blocks are leased and released by subprograms. When allocated to the subprogram, a block may be used as a temporary work area for a particular instance of the subprogram's execution or as a storage area for data to be queued to a subprogram or device. Dynamic block allocation eliminates some of the inherent duplication in assigning separate work areas for each subprogram and increases the value of re-entrant subprograms.

Lines. A line is a section of storage used to identify communication paths to subprogram and device queues. A limited number of lines are associated with each subprogram and device; data to be communicated must reside in a general storage block. A subprogram can transfer control of an allocated block of storage to a subprogram or device queue via an appropriate line. Lines are reserved and canceled by subprograms.

avoiding mutual lockout

The control program ensures that storage requests are made in a consistent manner, thereby guarding against the possibility of mutual suspensions. For example, if Subprogram A requests all of one type of storage and Subprogram B requests all of another type, both subprograms could be permanently suspended if each requested storage of the type currently allocated to the other. To avoid this, the control program ensures that lines are requested before blocks, which in turn must be requested before areas. If a particular type of storage is allocated to a subprogram, additional storage of the same type cannot be requested by the subprogram until all of that type storage is appropriately released or unlocked. A violation of these conventions is treated in the same way as a program interruption error, and execution is terminated. When storage is properly requested and is available in the amount requested, the control program returns control of the Computing Element to the requesting subprogram. If the desired storage resources are not available, the control program suspends execution of the requesting subprogram until they become available; it also reschedules the Computing Element that had been executing the requesting subprogram.

timing analysis

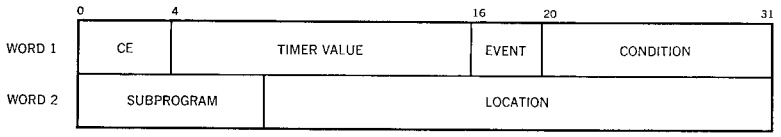
At an early stage in the design of the control program, a decision was made to incorporate a trace feature at critical points in the control program logic. Because the feature has proved to be significant in evaluating and checking multisystem operation, a description of it and some of the results obtained are mentioned.

The trace procedure generates timing analysis records (TAR's) that reflect many of the critical events and actions occurring during system operation. The TAR's are assembled sequentially in a buffer; when full, the buffer is recorded on magnetic tape. Two or more buffers are provided to allow TAR assembly and recording to proceed concurrently. (In the event that too little buffer storage is available, overflow counts are updated in the header of the last-filled buffer to reflect the number of TAR's per Computing Element that were not recorded. No attempt is made to delay control program operation because of inadequate buffer storage.)

Before the first TAR is stored in an empty buffer, the overflow counts are reset to zero and the elapsed time since startup/startover is stored in the buffer header. The elapsed time is measured in units of 1/2 second. In the event of buffer overflow, the elapsed time allows proper alignment of recorded buffers with regard to time.

The data in a TAR includes an event code, a Computing Element identity, and the identity of the time interval during which the event occurred. The time interval is obtained from a designated interval timer that is reset every 1/2 second to cause a timer interruption every 1/2 second. The units of the interval timer are 1/300 second with resolution of 1/60 second (i.e., the value in the timer is reduced by 5 every 1/60 second). Depending upon the type of event, additional data in a TAR may include a subprogram identity, a storage address, and an interruption code. Regardless of event type, each TAR requires two words (eight bytes) of buffer storage.

Figure 4 Timing analysis record format



CE COMPUTING ELEMENT

The format of the double-word TAR, shown in Figure 4, also provides a 4-bit field to identify the event type by a hexadecimal code (0 through F). Since TAR generation involves additional control-program overhead of approximately 150 microseconds per TAR, generation is placed under the control of a 16-bit mask. If the mask bit corresponding to an event code is set to 1, a TAR is generated for that event type. The setting of the mask may be changed at operator request.

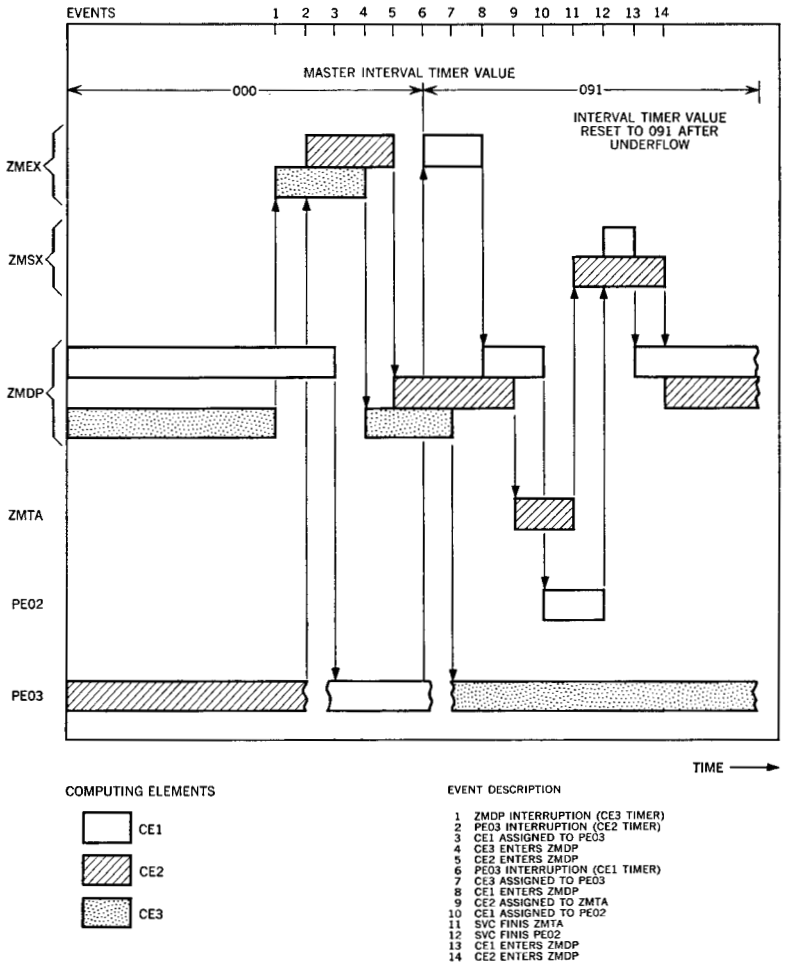
Since a series of TAR's represents a sequence of events with respect to time, it is possible not only to trace the activity of Computing Elements, channels, and devices but to examine the interactions of subprogram executions. Figure 5 shows an actual trace (developed from recorded TAR's) of a small portion of an integration test of the control program. The purpose of the test was to determine whether the control program could effectively control the activities of three Computing Elements. The subprograms represented in the trace are as follows:

ZMEX	External interruption routine of the control program
ZMSX	Supervisor-call interruption routine of the control program
ZMDP	Dispatcher of the control program
ZMTA	A subprogram that schedules other subprograms on a periodic basis and is scheduled every 1/2 second by the external interruption routine
PE02	A one-instruction (SVC FINIS) subprogram scheduled by ZMTA every second
PE03	A test subprogram with a permanent loop

The trace verifies that the control program performed correctly during the integration test. No more than one Computing Element was ever assigned to PE03 at any time, and an idle Computing Element was always immediately assigned to resume PE03 processing following a Computing Element's external interruption from its code. The re-entrant capabilities of the control program were also verified.

In addition to their use as a checkout aid, TAR's provide a basis for evaluating system performance. Since each TAR records the time interval during which the indicated event occurred, a sequence of TAR's reveal (to the nearest 1/60 second) the execution time of each identified operation bounded by two TAR's in the recorded sequence. From the TAR sequence, an off-line processing unit can

Figure 5 Trace of integration test



therefore calculate such statistics as: the average subprogram execution and delay time, the percentage of time that channels and devices are utilized, and the Computing Element idle time. Although the clocking resolution limits the accuracy of each subprogram execution time, the average execution times determined from large samples will normally be very accurate.

summary

In managing systems resources, the control program schedules operations in such a way that an idle Computing Element can continue the execution of a subprogram where an interrupted Computing Element left off. Storage resources in the form of lines, blocks, and areas are dynamically allocated to permit concurrent operation of subprograms without mutual interference. Because the control program is re-entrant, it can be entered at any time by any Computing Element via the interruption mechanism of the 9020 system. The control program coordinates the Operational Error Analysis Program and all application programs.

Timing analysis records generated by the control program provide a useful tool for system evaluation and checkout. The data provided by these records permit detailed time traces of the system's operation and performance statistics.

CITED REFERENCES

1. *System Description of National Airspace System Enroute Stage A*, Federal Aviation Administration, Washington, D. C. (April 1965).
2. M. E. Conway, "Multiprocessor system design," AFIPS Conference, *Proceedings of the Fall Joint Computer Conference* **24,** 139–146, Spartan Books, Washington, D. C. (1963).